# The Interchange of Haptic Information

White Paper Submitted by Novint Technologies, Inc.

## Overview

Haptic interaction is applied to and important for a wide variety of fields. There is active research and development, for example, utilizing haptics in the CAD/CAM, industrial design, CGA, geosciences and medical fields. There is, however, no widely used interchange format for haptics-relevant information. More widespread research and development efforts utilizing haptics are, of course, relatively recent so the lack of widely used interchange formats is to be expected. Yet this lack of haptics information interchange formats is an impediment to the development of the field. It is difficult to communicate both "finished content" and methods/approaches. Starting to understand the issues associated with the interchange of haptic information and defining an approach to achieving this interchange could be very useful to the field. Although it is quite likely that any early attempts will be shown to be incomplete or ill directed over time, it is only by attempting to address these issues and learning from the process that commonly used haptic interchange formats will emerge.

This "white paper" discusses some of the issues related to the interchange of haptic information, outlines one possible framework for an interchange format and discusses a particular instantiation of this framework. The authors wish to emphasize that this is “work in progress”. The main goal of this paper is help to start to understand the scope of the issue and to generate meaningful discussion concerning the interchange of haptic information.

## Interchange Format Standardization Background

An important factor in the success of any technological or scientific field is the evolution and development of methods to help specify and communicate core ideas and concepts within that field. In other words, some form of common “language” or notation to capture and convey key observations, notions, models, theories and other relevant information is advantageous. In reality, there is usually a collection of languages/notations/grammars that are utilized to help support research, development and utilization within a given field.

The interchange of “scene” information (i.e., objects and their characteristics) in computer graphics/animation (CGA), CAD/CAM and virtual environments is one important example of the use of notation and grammar to specify and communicate technological information. Geometric and visual rendering information is communicated via commonly accepted file formats. This allows compact interchange of scene information between users and, oftentimes, the utilization of this data across applications. Benefits include: a) support for scene persistence; b) tools created by different organizations can be used to refine and render the scene; and c) representational models and characteristics are made explicit.

Graphical interchange formats for three-dimensional data include proprietary and vendor-derived formats. Many of these have such widespread use that they have become "de-facto" standards.

These include widely used vendor-specified formats such as Alias Wavefront (i.e., *.obj), 3D Studio (i.e., *.3ds) and AutoCAD (i.e., *.dxf). Other graphical interchange formats are the result of actions by standards committees in defining new industry-wide formats or refining vendor-specified formats into more "vendor neutral" forms. Examples of such standards in three-dimensional computer graphics, animation and CAD/CAM include IGES, VRML 1.0/97/2.0 and, more recently, X3D. All of these file formats allow the specification of three-dimensional geometry. Most of these file formats allow specification of visual rendering relevant parameters such as lighting, color and texture. VRML and X3D also allow hyperlink and behavioral information to be encoded.

As can be seen, even a relatively mature field such as computer graphics has a plethora of vendor-specified and "standards based" information interchange formats. The most successful of these tend to be "de-facto" standards or are based on de-facto standards. This is a natural consequence of the maturation of a field. Initially, since no other alternatives exist, specific vendors or research groups develop interchange methods for their own use. The utility of these interchange formats is assured as the formats are refined and extended within the context of real applications of relevance to the field. As the programs that utilize these methods become more widely used, the particular interchange format becomes of interest to other vendors and research organizations. Typically these other groups will directly utilize or provide conversion utilities to the interchange format in order to reach the community that is using the interchange format. In time, the interchange format may be the basis for (or at least an important component in) industry/field "standards" efforts. Typically, the more "open" that the original interchange formats are, the more quickly that they evolve, the more quickly that they are adopted and the more widely they are used by the community.

Competing interchange formats and standards, of course, emerge. This tends to be healthy for the field in that different interchange formats oftentimes tend to address different/more specific needs in the field. They also tend to be informed by or build on previous efforts so that refinement and evolution can occur relatively rapidly. It is sub optimal, however, if these competing interchange formats tend to address the same needs and requirements. Sometimes standards efforts are undertaken when there is too much overlap and fragmentation in interchange formats.

# Haptics Information Interchange Requirements

What kind of information is required to support haptic interaction? The answer to this question depends upon the overall application domain, the characteristics and limitations of haptic devices, the methods used to control these devices and human perceptual “haptic percepts”.

## Field of Use Issues

Different fields and applications emphasize different aspects of haptic rendering and interaction technology. Many manufacturing CAD/CAM applications, for example, would be well served with the ability to haptically render rigid solids and to spatially manipulate them. Industrial design applications tend to put more of a premium on rendering of surface properties such as compliance and texture. The rendering of deformable bodies is important to many applications in the medical domain. Even among these few example domains, a variety of underlying geometric models are encountered -- both surface and volumetric representations. Applications and domains with more abstract data, such as computational fluid dynamics and geoscientific data

processing, do not even utilize classical geometric models. These domains tend to emphasize haptic rendering and interaction with field data and abstract/derived/computed spatial properties. Finally, other domains, such as CGA, utilize and interchange information related to the dynamic behavior of objects within scenes.

## Device and Control Issues

Data interchange formats and approaches are also informed and constrained by the rendering models/approaches and technologies that are used. In computer graphics, for example, scene geometry is often supplemented by information to support widely used visual rendering approaches. Color information (often using a multi-component model) is provided along with lighting information. It should be noted, that like geometric models, even "simple" information such as color can be specified in a variety of formats (e.g., RGBA, CMY, etc.). These options exist even in a field where the rendering technologies and devices are relatively mature and "standardized" (e.g., CRT displays and LCD displays).

In the haptic rendering and interaction domain, the devices and the underlying control methodologies/approaches are anything but standardized. For example, the number of degrees-of-freedom supported by haptic interaction devices for "kinesthetic feedback" can range from one or two degrees of freedom for more "gaming oriented" devices (e.g., Logitech force feedback mouse) to dozens of degrees of freedom for some special purpose and research devices (e.g., SARCOS dexterous arm master). Control methodologies range from simple "open loop" position/force control approaches (typically utilized with devices such as the PHANTOM haptic interface device) to "closed loop" bi-lateral servo control systems where forces are directly sensed and controlled (e.g., SARCOS dexterous arm master).

Tactile feedback devices or components come in at least as many configurations as their kinesthetic feedback cousins. These devices may be able to be roughly categorized by such characteristics as area of influence and density of "stimulator" elements but there is little standardization beyond that. For example, some devices utilize electro-cutaneous stimulation techniques and others utilize mechanical vibration or deformation techniques. So even the underlying manipulated state variables are different.

## Perceptual Issues

Perceptual models also enter into what kind of interchange information is required for haptics. The sensory capabilities of our visual systems, for example, are reflected in some color representation techniques used in CGA. Although much is known concerning human haptic capabilities, a commonly accepted haptic "perceptual language" has yet to emerge. Very broad haptic discrimination categories such as shape, compliance and texture, however, can help inform the development of a haptic information interchange approach.

# Other Efforts

A significant percentage of software research and development efforts utilizing haptics define or otherwise use some representation of the visual and haptic attributes of the objects that they employ. Whether the representation is implicit or explicit, some representation of the haptic properties of objects is used in order to support scene and program persistence. To date,

however, little of this work has entered the public domain. For the most part, these efforts are project/institution specific or proprietary. An example of a proprietary approach may be found in the ReachIn Technologies Magma API. Specific extensions to the Virtual Reality Modeling Language are used in Magma to capture haptic information in a form that is consistent with the haptic rendering approach used by that company. The details of the haptic rendering algorithms themselves are not in the public domain. There is some initial activity, however, in the public standards sector. CSIRO has proposed extensions to the MPEG-4 standard. These extensions, however, are based on Magma work and suffer some of the same limitations – they assume and "prescribe" a specific set of rendering capabilities but do not specify the details of the algorithms themselves.

# Haptic Information Interchange Framework

Given the state of haptics research and development, is it even meaningful to discuss haptic information exchange methods and formats that could be widely applicable? Can we do more than simply specifying methods for capturing haptic-relevant control variables and information for use by proprietary or highly simplified algorithms?

We believe that the answer to these questions is yes. Our approach is to go beyond "prescriptive" approaches and develop an extensible descriptive framework that maps well to existing haptic rendering approaches while maintaining structural flexibility to grow over time.

The key aspects of the haptic rendering and interaction domain that must be taken into account in the haptic information interchange framework are:

- ?? Model of the Haptic Rendering Process
- ?? Spatial and Behavior Data to be Rendered
- ?? Model of the Haptic Device and Associated Control Mechanisms

The remainder of this paper provides an overview of a particular haptic information interchange framework and discusses a specific implementation of that framework utilizing X3D.

## Modular Haptic Rendering Chain

A fundamental observation in developing haptic information interchange framework is that haptic rendering of a wide variety of data types and behaviors can be modeled as a connected series of processing steps or modules. Previous processing modules in the haptic rendering chain potentially inform each module in the chain. Any given module may also affect the data.

For example, in allowing haptic rendering and interaction with a dynamic rigid body, one typical approach is to compute the location of the surface relative to the haptic interaction point and use this information to compute surface compliance (i.e., a force in the direction of the surface normal). When surface texture is computed, the direction of the compliance force can be used to filter texture information so that it is to be applied tangent to the point of intersection. To compute dynamic behavior for the object, the results of the previous computation steps are used as the basis for computations that alter the position and orientation of the object. Figure 1 shows this particular rendering chain.

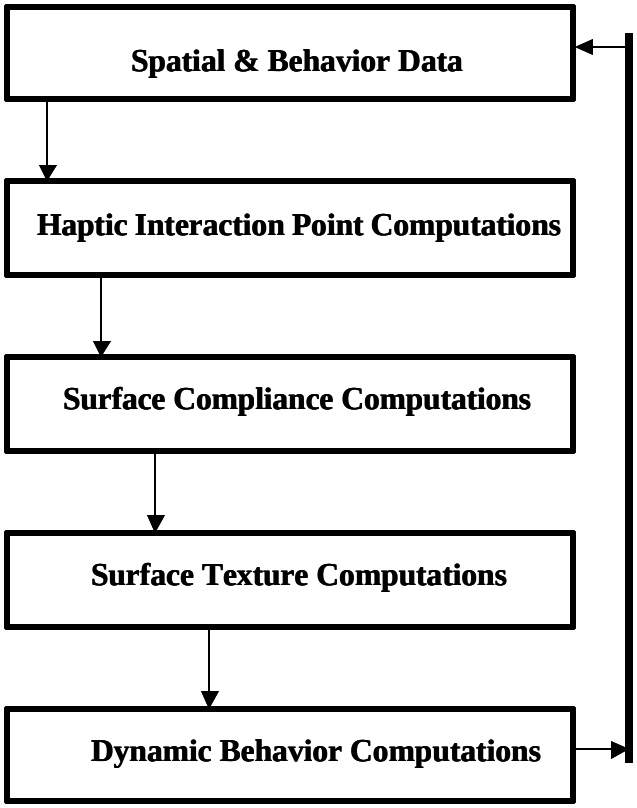


**Figure 1 – Haptic Rendering for Dynamic Rigid Body**

A more abstract representation of this "modular haptic rendering chain" (MHRC) is given in Figure 2. Since the actual algorithms utilized in any given step or module can vary within and across domains, the framework is designed to allow module specific rendering code to be specified in the interchange format itself. In other words, allowing the interchange format to provide the actual rendering code for modules supports portability and extensibility.

In addition, the MHRC framework allows for specification/designation of module algorithms by named reference. This allows module actions to be specified by referencing a commonly accepted/available rendering algorithm (as these become more common). This same mechanism can be used to reference more proprietary approaches. As the field matures, it is expected that the key rendering steps for given domains will become more "standardized" and that the framework could specify a default "common" algorithm to use as well as a "proprietary" algorithm to use when it is available.

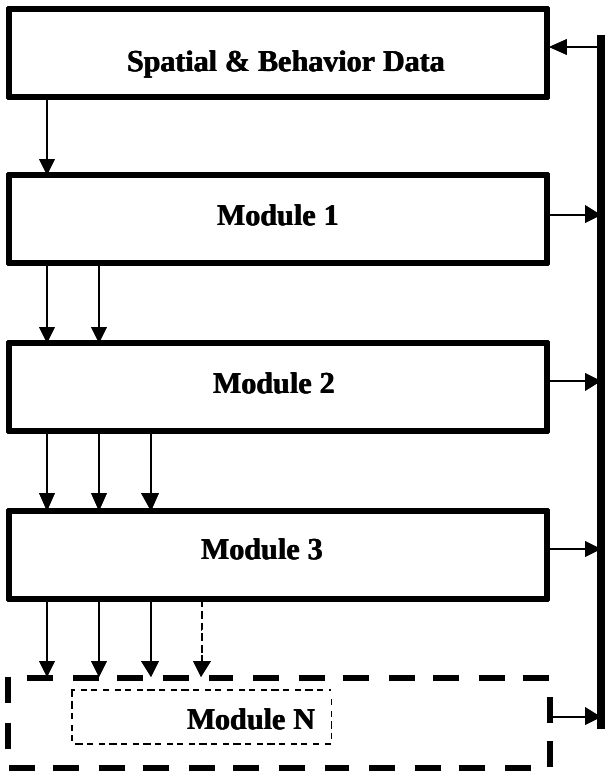


**Figure 2 – General Modular Haptic Rendering Chain**

The MHRC approach not only allows for abstraction of the haptic rendering chain but it also provides a way to deal with issues associated with different haptic device capabilities and control mechanisms. If information up and down the module chain is viewed as state vectors then varying degrees of freedom can be efficiently encoded. With proper abstraction, this mechanism might allow devices with differing degrees of freedom to use the same module chain. Devices with fewer degrees of freedom could just ignore non-relevant parts of the state vector used in carrying "computed forces". For example, a device that has only three positional degrees of freedom could ignore force information relating to rotational degrees of freedom. At this time, different control approaches are envisioned to be handled within the modules themselves.

The spatial and behavior data that is encapsulated in the MHRC framework, of course, domain and applications specific. Some applications, for example, will utilize polygonal data, others rational B-splines and yet others will utilize volumetric data. The MHRC framework allows this data to be encoded in the interchange specification. Initially, different rendering chains will be used for different data types. It is felt that in time, however, more general haptic rendering techniques can be developed and all that may be required to utilize them is a "data pre-processing" step.

## A Test Implementation


**Figure 3 – The Layout Application**

Clearly there is significant clarification and refinement that must occur for the MHRC framework to prove its utility. As a first step and in order to provide a concrete test of the overall concepts, a dynamic rigid body application utilizing the MHRC approach is being developed. This application, known as Layout (Figure 3), allows rigid bodies, modeled as polygonal data to be instantiated. These objects exhibit surface compliance and texture. These objects can be haptically manipulated and exhibit dynamic behavior in the form of physically based simulation of rigid body motion. Constraints can also be placed on object motion. In order to test the feasibility of utilizing and building on existing graphical interchange standards, the MHRC interchange format instantiation is based on the X3D standard. The X3D standard essentially captures the VRML specification within an XML syntactic framework. XML is extensible so we were able to "piggy back" the MHRC components using X3D itself. This was done in such a way that a haptically enabled scene can still be viewed using a standard X3D "graphics only" viewer. In addition, XML and X3D allow us to specify MHRC module algorithms directly. Appendix 1 provides additional specific information on the Layout test implementation of the MHRC framework

# Summary

The specification of a method and approach to the interchange of haptic information is a challenging endeavor. The relatively early state of the haptic rendering and interaction field, coupled with a wide variation in the types of data that must be rendered and the devices available to render them, makes the specification of a "prescriptive" haptic information interchange format sub optimal. We have proposed a way to start to organize and specify haptic rendering and interaction known as the Modular Haptic Rendering Chain (MHRC). This approach is "descriptive" in nature – it provides a framework for embedding a variety of haptic rendering approaches in a structurally coherent manner. In order to better understand and test the MHRC concept, a haptically enabled dynamic rigid body application, known as Layout, has been developed and is being refined. Although the MHRC approach is clearly imperfect and requires significant refinement, we believe that it is a solid "stake in the ground" and bears further exploration.

# Appendix 1 – Test Implementation

## MHRC in a Dynamic Rigid Body Application

The Layout application provides an initial test implementation of the Modular Haptic Rendering Chain (MHRC) framework. As previously discussed, this application allows rigid bodies, modeled as polygonal data to be instantiated. Haptic rendering of texture and surface compliance are supported. Objects can be haptically manipulated and exhibit dynamic behavior in the form of physically based simulation of rigid body motion. Constraints can also be placed on object motion. For this application, the MHRC framework is instantiated using X3D formalisms. Parsing of algorithms for MHRC framework modules has not yet been added. This particular instantiation of the MHRC framework is referred to as the Multimodal File Format (MMFF) in this document.

# X3D Encoding

## X3D Features Used

The minimum set of required X3D nodes is:

- ?? Transform
- ?? Group
- ?? Geometry
- ?? Shape
- ?? IndexedFaceSet
- ?? Appearance
- ?? Material
- ?? Color
- ?? Coordinate

In addition, support for prototyping is needed. Other node types and X3D functionality can be included in the file, and should be handled but do not need to be implemented. The interactive aspects of X3D are not (initially) needed. These should be parsed by an application, but may be ignored.

X3D content can be authored in a variety of data encoding formats, including XML encoding format and VRML encoding format. The examples throughout this document use the VRML encoding format, since it's easier to read. In practice, however, the Multimodal File Format uses the XML encoding format. The XML encoding of the examples used in this document are provided in digital format along with this document. These files were generated using the X3D-Edit program provided by the X3D Consortium. The current (16 February 2002) version of X3D Document Type Definition (DTD*) x3d-compact.dtd* was used.

## Object

An Object node will describe each object within the scene. The Object node will specify:

- ?? Position
- ?? Orientation
- ?? Geometry
- ?? Graphic appearance
- ?? Haptic Properties
- ?? Behaviors
- ?? Bounds

## Appearance, Transform and Bounds

Nodes to describe appearance, transform information, and a bounding box are already in X3D, and we continue to use them within the Object description. The Object will extend the X3D Shape node:

```
Object {
        position
        orientation
        geometry
        appearance
        behaviors[]
        bounds
}
```

The HapticAppearance node will extend the X3D Appearance node by adding a surface field, which is filled by a Surface node:

```
HapticAppearance
{
        material
        texture
        texture_transform
        surface
}
```

The Surface node is described in the next section.

## Haptic Properties

A Surface node will specify the forces generated while touching the object. It is composed of a Compliance node and a list of Force Modifiers.

The Compliance node will specify the algorithm used to calculate the initial surface contact force. A default SpringForce node will be provided, which will generate the force using a spring model and the tools distance from the object surface. A different force algorithm can be applied by setting a different Compliance node.

The Force Modifiers filter the initial force, to specify additional haptic properties of the surface (e.g., friction, texture maps, etc.). Modifiers will take as inputs the Object's state, the initial force, and the force after all previous modifiers have been applied. Each modifier will provide fields for all of the additional information needed to describe it. If two modifiers added to a surface have the same fields, the information will still be specified in each Modifier. For example, many Modifiers may have a spring constant field but it will still be specified in each Modifier.

### *Ordering*

The order in which Modifiers are applied may affect the output. Two levels of ordering are provided. Each Modifier will have a priority field indicating the order in which it is applied. Modifiers with the same priority are applied in the order they appear in the file. Each Modifier type will have a default priority, to specify the preferred ordering; the priority can be changed by filling in the order field. It is expected that for most cases the default order will suffice.

Priorities of 1-10 are allowed. Any Modifier that doesn't require an input force will have a priority of 1. Modifiers that should always be applied last will have a priority of 10.

An example Surface:

```
Surface
{
        compliance: Spring {     stiffness 500 }

        modifers [
                BumpMap{
                        texture: {....}
                        height: 0.5
                        priority: 2
                }

                Damping {
                        ratio: 0.2
                        priority: 10
                }
        ]

}
```

## Behaviors

Behaviors modify the Object's state, but do not calculate any forces. They may take the final force calculated from Surface as an input. Dynamics, and Constraints will be specified as behaviors. For example:

```
Object{
        behaviors[
                PlaneConstraint{
                        normal 0 1 0
                        distance 0.5
                }
        ]
}
```

# X3D Integration

Several additional node types have been specified. X3D allows new node types to be specified using the PROTO statement. Each prototype is derived from a parent type, and the prototype can be used in place of a node of the parent type.

PROTO declarations for the extra node types specified are shown in figure X. (haptic.wrl). The Object node is constructed from a Shape node, with an integrated transform to provide position, orientation and bounds. A HapticAppearance node extends the Appearance node by adding a field for a Surface node. The Surface node is a child node, and contains fields as described earlier. Behavior nodes subclass Child Node, so they can be added to a Group.

The relationship between Object, Surface, Modifiers and Behaviors will not be described in the file with X3D routes, it is implied by the node types.

## Reading From a Standard X3D browser

Each of the new nodes, including each type of Modifier and Behavior, will be defined with a PROTO statement. All of these prototypes can be collected in a single file, each and can then be referenced from data files using the EXTERNPROTO statement. This allows the data file to be parsed by a standard X3D browser. The new node types will be "inert" when read into a standard browser.

A short example.

*haptic.wrl*

```
PROTO Object{…}
PROTO HapticAppearance{…}
```

*data.wrl*

```
EXTERNPROTO Object "Haptic.wrl#Object"
EXTERNPROTO HapticAppearance "Haptic.wrl#HapticAppearance"

Object{
        appearanceHapticAppearance {}
}
```

# Extensions

The X3D spec allows for PROTO nodes to be implemented in an extension (i.e., programming) language and included in the scene.

The haptic renderer could be extended to support this capacity for Modifiers. New modifiers could be implemented as a Java™ class, and added to the scene through X3D's externproto syntax. For example:

```
EXTERNPROTO MyModifier "MyModifier.class"
```

In this way new surface Modifiers can be implemented and used with no need to update the haptic renderer. New behaviors could be implemented in the same way.

The Java™ classes for the new modifiers would be distributed with any scene file that used them.

# Example

As an illustrative example, two files, in VRML format, are provided. The *haptic.wrl* file contains all the Proto definitions that are needed to allow a standard X3D browser to parse the files.

The *sample.wrl* file contains a scene with a single Object. The data file (i.e., *sample.wrl*) uses ExternProto nodes to get the definitions of each new node type from the *haptic.wrl* file. A sphere Object is specified. Two "force filters", a BumpMap and a Force Anchor, are added to the surface forces. A constraint and gravity are added to the behaviors

## Haptic.wrl

```
PROTO PlaneConstraint{
        field SFFloat distance 0
        field SFVec3f normal 0 1 0
        field SFBool magnet false
}

PROTO Surface {
        field SFNODE compliance
        field MFNODE modifers
}

PROTO HapticAppearance{
{
        DEF APP Appearance {}
        EXTERN APP     # Export causes HapticAppearance to be derived
                                from Appearance

        field SFNODE surface
}

PROTO Object
{
        field SFVec3F position
        field SFRotation orientation
        field MFNODE behaviours
        field SFFloat scale

        DEF TRAN Transform
        {
                children DEF SHP Shape { }
        }

        position TO TRAN.position          # Route fields
        orientation TO TRAN.rotation

        EXPORT SHP   #Export causes Object to be derived from Shape
}
```

Other PROTO's for Modifiers would also be specified in this file.

## Sample.wrl

```
#include prototypes from haptic.wrl file.
EXTERNPROTO Object [] "http://.../haptic.wrl#Object"
EXTERNPROTO HapticAppearance[] "http://.../haptic.wrl# HapticAppearance "
EXTERNPROTO PlaneConstraint[] "http://.../haptic.wrl# PlaneConstraint "
EXTERNPROTO ExternalForce [] "http://.../haptic.wrl# ExternalForce "
EXTERNPROTO BumpMap [] "http://.../haptic.wrl# BumpMap "
EXTERNPROTO ExternalForce [] "http://.../haptic.wrl# ExternalForce "

# Add an object

DEF MY_SPHERE Object {
        position 0 1 0

        apearance HapticAppearance {
                geometry Sphere {}

                material Material{}

                surface Surface {
                        compliance SpringForce {stiffness 500 }

                        modifiers [
                                BumpMap {
                                        texture ImageTexture {}
                                        height 0.5
                                        initial_force TRUE
                                }

                                AnchorSet {
                                        anchors AnchorPoint3D {
                                                forceToMove 5
                                                }
                                }
                        ]
                ]
        }  # end appearance

        behaviours[
                PlaneConstraint {
                        normal 0 1 0
                        distance 0.5
                }

                DEF GRAVITY ExternalForce {
                        force 0 -10 0
                }

        ]
}
```